

Noble Ape Simulation

Tom Barbalet

Noble Ape

Building on my early ideas about biological and cognitive simulation, I began developing the Noble Ape Simulation in June 1996. I was nineteen then and going to a university in Australia. I was captivated by the speed of software propagation and thought a small simulation would not only test the ideas but disseminate them quickly to a wide variety of people over the Internet.

Developed with optimization and access as its central tenets, the simulation aimed originally to

- simulate a biologically diverse landscape (initially tropical islands) and
- simulate the ape inhabitants' cognitive processes.

The simulation creates a rich ecological environment over a relatively large area through which the simulated-sentient noble apes can wander. The user interacts with the environment through a number of simultaneously displayed graphical views.

Developing the simulation

The Noble Ape Simulation includes two quite distinct developments: the simulation core and the simulation graphics. Both are modular (see Figure 1) and both regularly support other developments aside from the Noble Ape Simulation (see <http://www.nobleape.com/sim>). These two developments also contain a number of smaller modular components.

The simulation core contains a landform generator, a weather and season generator, and two simulations: biological and cognitive. In addition, the simulation core contains a set of optimized functions that includes a random-number generator, an optimized Bresenham function (used for line-of-sight and line drawing), and a file-protocol parser.

The biological simulation relies on quantum mechanics. The landscape is a wave function whose operators give properties such as surface area, height, and current sunlight intensity. Certain plants like certain characteristics—lots of sunlight and flat land, for example. The simulation describes these characteristics through the quantum mechanics operators on the land that provide numerical representations of “sunlight” and “flat land.” It thus describes the plant and animal population densities by the quantum mechanics operators on the landscape (wave function). Last year, I expanded the simulation to include a much larger environment. Although the quan-

tum mechanics simulation primitives remain, the simulation now covers a number of large and complex landscapes—not just tropical islands (see Figure 2).

The cognitive simulation relies on two competing 3D mathematical effects to describe two basic cognitive effects—fear and desire. Fear represents instantaneous reactions and desire represents future goals. These two appeared to be the most fundamental elements to model in a cognitive simulation.

The two competing mathematical effects used to represent fear and desire came through my attempts to simulate primitive population growth in 3D simulated “agar” in the mid-1990s. Through very applied means, this showed me two competing mathematical equations, one reactive to change (used to represent fear in the cognitive simulation) and the other resistive to change that produced a distinct recalling-memory effect (used to represent desires). Into this cognitive agar, I fed information that should motivate movement. Primarily simulated visual information, these additions also included some early experimentation with simulated audio information. The cognitive simulation is still a work in progress.

Thanks to the Noble Ape Simulation, others have used, and continue to contact me with, their own cognitive simulation metrics. The Noble Ape simulation gives them an environment for testing their cognitive simulation methods. (See “The Cognitive Simulation” sidebar for further information about the mathematics involved.)

In developing the simulation, I wanted to explore, in particular, the possibility of simulating a biological environment, simulate cognitive processes, and more broadly, gain international recognition and collaboration for the development. How far could it go?

To speed the software propagation, I placed two constraints on the development:

- the total executable must be under 100 K, and
- the software couldn't favor particular hardware.

This latter point was important as the simulation currently runs on Windows, Macintosh OS, Linux, and a number of Unix variants through X Windows. All graphics generation occurs internally, within the simulation executable.

The simulation development has been pragmatic and relatively isolated. Almost all new development starts from mathematical primitives, so I rarely use external sources. The development's broad nature and my age

The Cognitive Simulation

The ape's brain describes information transfer. Think of it as an information transfer medium you can feed with visual and audio information to produce movement. Additional outputs can be added, too. For speed and simplicity, I set the original size of the brain at 32 cubed. Early work around larger brain sizes showed that most of the effects required could be found in the 32 cubed brain size (although more interesting effects came into play with brain sizes from 128 cubed on).

The simulation abstracts the mathematical properties of fear and desire by combining two-time dependent 3D equations. Assuming that the following equation represents a point in the ape's brain:

$$b(x, y, z, t + 1) = a \forall l + b(t) \forall m + (b(t) - b(t-1)) \forall n;$$

Where x , y , and z are brain cell coordinates, t is time, and

$$a = b(x-1, y, z) + b(x, y-1, z) + b(x, y, z-1) + b(x+1, y, z) + b(x, y+1, z) + b(x, y, z+1).$$

Fear is governed by the n value, Desire is governed by the l value, and m is a normalization constant. In the current simulation, these values change when the ape is awake or asleep. Early tests relating to blood sugar and panic produced interesting results. In general, the panic condition is set by substantial effects of fear.

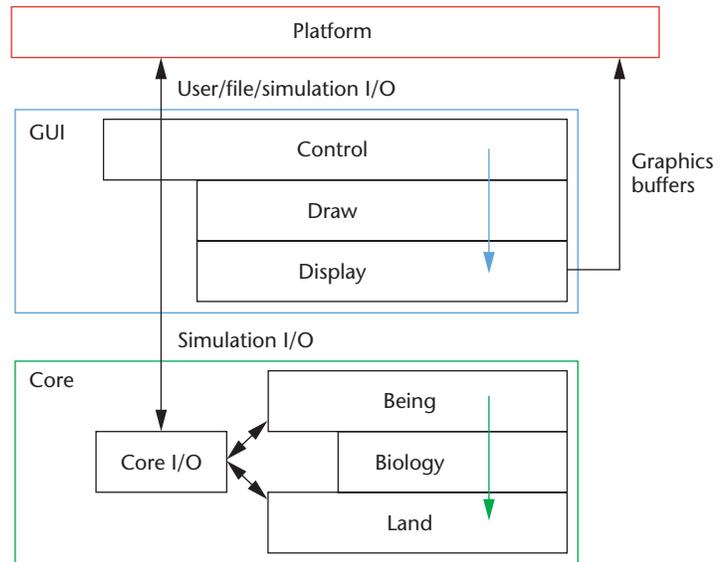
While the current simulation feeds the brain with semirandom stimuli, the most productive method has placed fixed single-point or small surface transmission ($2 \forall 2 \forall 1$ or $4 \forall 4 \forall 1$) sensory neurones into the brain. These feed information relating to visual or audio cues.

I added motor neurones to the brain to produce reactive movement. The current simulation is a hybrid between the idealistic brain simulation described here and survival simulation. In the future, the cognitive simulation ideally will

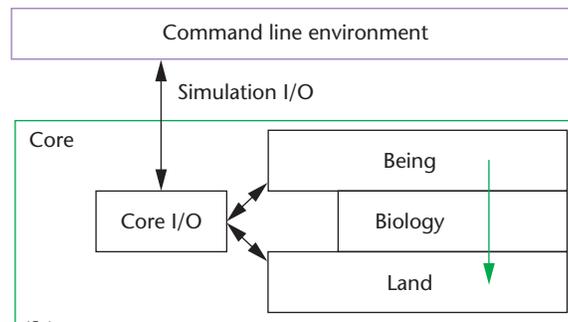
made it difficult to get formal university involvement, so I developed it independently, although a number of academics participated and used the early simulation.

From freely available to Open Source

Written primarily in C, the simulation's source code has always been available, even though Open Source is a relatively contemporary concept. The Open Source definition has only existed for roughly half of the Noble Ape development. Although the GNU and BSD licenses are much older, the idea of putting a license with the simulation—particularly a license linked to a US corporation—didn't seem logical when the development

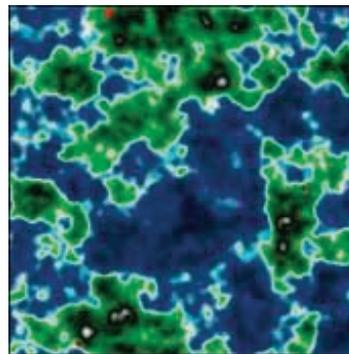


(a)



(b)

1 The simulation is extremely modular: (a) Ocelot interface; (b) command line environment



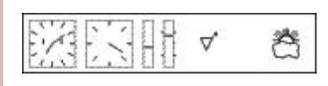
2 Simulated landscapes contain a variety of land features.

was coming from an Australian shed in 1997. Then, it appeared that software licensing was part of a corporate legal addition to the perfectly natural pursuit of making your software and source code freely available. Licensing seemed a paradox, both practically and ethically. After all, software licenses let large companies gain astonishing rights with regard to defective software. In many ways, this idealistic ethos typified the initial development of Noble Ape. (See the "Accessing Information" sidebar for more information about the source code.)

From this initial idealism, the only direction was down.

Accessing Information

I designed the Noble Ape Simulation to be interrogated at a source code level (see Figure A). The challenge has been providing information for users who don't need or know how to access the source code. The current file format provides some of the simulation's inner workings. Table A indicates what is in the simulation and what can be accessed through its current file format versus what can only be accessed through the simulation source code.



A Meters display numerous parameters.

Table A. The Noble Ape Simulation: Source Code or File Format

Feature	Source or file
Land Information	
Physics	Source
Landscape	Source (File planned)
Date and time	Source or File
Weather simulation method	Source
Weather conditions	Source or File
Biological simulation method	Source
Biological Information	Source (File planned)
Being Information	
Movement simulation method	Source
Location and direction facing	Source or File
Sex and DNA information	Source or File
Energy usage	Source or File
Current speed	Source or File
Cognitive Simulation	
Cognitive simulation method	Source
Cognitive constants	Source or File (from 0.665)
Brain states	Source (File planned)

Within the development's first 18 months, a company had achieved substantial capitalization and publicity for a development strikingly similar to Noble Ape. An Australian Internet publication that covered the Noble Ape development contacted me about a quote running in a major popular science publication, which came from the simulation's manual but was attributed to this company.

That company had exploited aspects of the development without reference, but there was little I could do about it. I lived in a shed and missed meals so that I could afford a hosted domain name. Rather than being fatalistic, I saw that the development sections the company had taken were not developed or understood enough to yield anything more than fudged results.

Following the Internet magazine's request for a response, I contacted the company directly and offered to present my latest work, thinking that the potential of collaboration would be better than unreferenced exploitation. I received a cold response. By taking the documentation's words rather than the development's ideas, the company sealed its fate. They achieved substantial publicity and a substantial user base before eventually folding.

Once the early development was complete, I found commercial sponsorship for my independent R&D separate from Noble Ape. With this support, I could live and maintain the development, releasing the software and source code in the evenings. Rather than glamour, commercial sponsorship meant a basic stipend. My quality of life did not improve, and I continued to live in the shed. With my development priorities elsewhere, the simulation almost ground to a halt, although other aspects of Noble Ape continued.

Commercial sponsorship let me travel and meet company executives, academics, and individual developers.

On my most extensive trip in 1999, I met a journalist who had followed the Noble Ape development since its onset. He wrote an article on a particular aspect of the graphics from the development, but he wrote it without my knowledge and published it without fact checking. The article came out while I was traveling—away from email—and was syndicated in more than 20 publications worldwide. In a matter of hours, readers flooded my email inbox: The Noble Ape Web site received more than 40,000 hits an hour.

The response from the article let me return to the US and fulfill a lifelong dream of living in Silicon Valley. Before the syndicated article, I had been privileged to meet a number of my childhood technology heroes. Living in the Bay Area, I could spend more time with them. Through that period, I worked closely with a number of startups and regularly received pitches for business models or companies based around the Noble Ape development—all of which required me to give up between 50 to 70 percent of my interest in the development for relatively speculative ends. I was disinterested, to say the least. I arrived in Silicon Valley at the end of 1999, just before the collapse of the speculative technology industry (together with the dot-com collapse). I moved to the UK in 2001.

The Noble Ape development suffered during my stay in the US. Although the development is not extremely time-consuming, Noble Ape does require some time commitment, and my time was scarce. When I arrived in the US, the Noble Ape mailing list held roughly 1,400 names. When I left, it was down to six. While the simulation source code had been relaunched in 2000, the development was virtually dead.

In June 2001, stopping the development was an option. The decision came to a head in Stockholm where

I was traveling with my employer at the time. I took with me a laptop on loan, some Noble Ape music CDs, and the simulation's source code. Through this period, I decided to relaunch the source code formally in Open Source. I also totally rewrote the source code. Fortunately, I was in Stockholm long enough to almost fully complete the "Stockholm Re-Write" on location.

How would formal Open Source have protected me in 1997? Because much of the Open Source infrastructure was not available in 1997, I probably would not have received any additional protection. I have modified my behavior since then. I don't provide detailed future projection documentation online anymore. I keep the Noble Ape development in three- and six-month information cycles. Having said that, if a commercial development appeared that used the documentation or source code from Noble Ape without reference, there are now well-established legal avenues for Open Source developers.

Simulation graphics

I originally developed the simulation with operating system graphics primitives (see Figure 3). For example, when I first wrote the simulation for the Macintosh, I assumed that Apple's QuickDraw would be used. This let users draw lines with QuickDraw's MoveTo and LineTo tools. I also used QuickDraw to draw the weather icon, which represents time of day and weather condition. This approach worked because the 16-MHz initial development machine took so much time doing the internal simulation that the screen drawing looked quite nice.

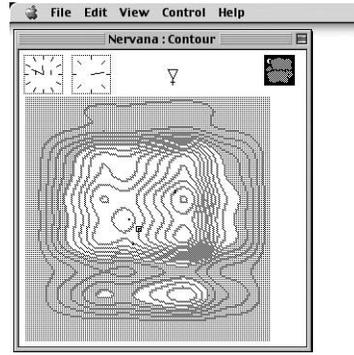
In porting the code to Windows in 1997, I modified it so that lines and other graphics the simulation used would be an OS-level primitive.

In 1997, the development received an equipment grant, which meant that I could develop Noble Ape in color. Over this period, I developed a first-person perspective engine for showing the simulation's rolling landscapes from the ape's eye view. This graphics development received considerable media interest in 1999, although I shelved it soon after.

When I dusted off the simulation and rereleased it in 2000, I reintroduced it on much faster machines. At roughly the same time, I started developing the Planet Noble Ape environment as the next stage of the Noble Ape simulation. Through the Planet development, it became evident that platform-specific graphics were the primary bottleneck for displaying planet movement in real time.

Rather than using hardware-assisted rendering, I developed internally optimized software rendering. This ethos, originally employed with the Planet development, moved back to the simulation. The graphics primitives comprised a pen state, erase window, drawing pixels, lines, and icons. The initial graphics were simple line vectors that survive in today's bottom-end porting. Maintaining these low-end graphics is critical, because they often provide the easiest porting route for new OS types and they are still fast.

In late 2001, I flirted briefly with an OpenGL port of the simulation. While this port worked and produced a color landscape environment, an alternative color land-



3 The early simulation was quite primitive.

scape engine clearly was needed to provide smooth, real-time graphics.

Ocelot

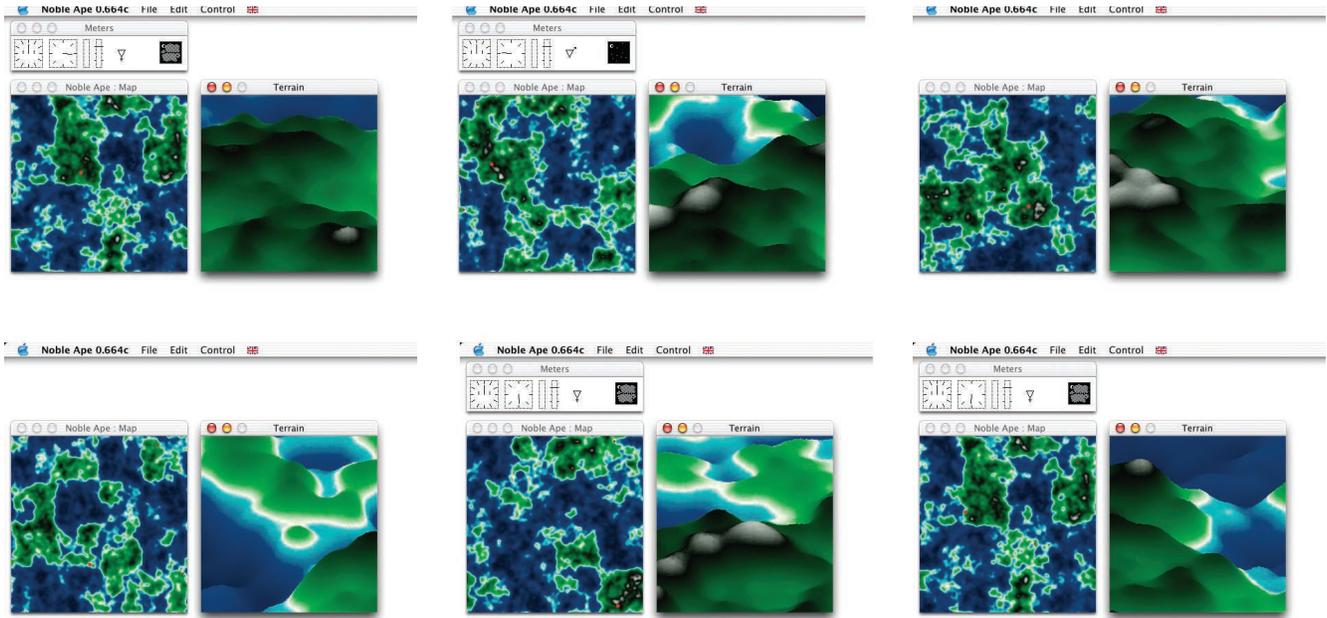
In mid-2002, I began work on an isometric projection landscape engine, aiming to provide smooth surfaces rendered from a third-person perspective (see Figure 4, next page). Operating in non-real-time, the initial engine was an ANSI C program that produced a raw data file. The output files were monochrome: high z -values were lighter than lower ones. From my initial ANSI C tests, I found I could render 15 frames per second on my 133-MHz machine—too slow for real-time use. I code-named the algorithm Ocelot.

The principle behind the algorithm was to render each pixel from the window's bottom to its top. The isometric projection meant that the points at $z = 0$ were known and rendering was a matter of following the dz along the vertical bands. The pixels were filled with a while loop. When a section dipped below the existing high point, the while loop simply fired false and the algorithm continued. The initial algorithm associated color with height, so the algorithm selected and interpolated the high beach, deep ocean, low grassland, and mountaintop colors, producing smooth color.

Over the next six weeks, I rewrote the algorithm so that eventually it could render 30 frames per second. As the main optimization, I removed all multiplication from the inner loop, except bilinear interpolation. The bilinear interpolation was faster than feeding in higher resolution memory maps. Memory accessing was typically the slowest part of the algorithm, so z -values were cached locally and changed only when needed.

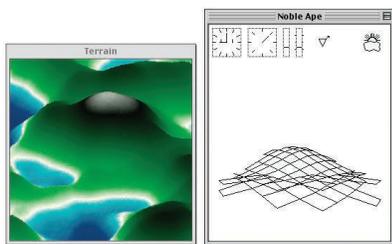
In terms of optimization, 30 frames per second was not shabby, but it wasn't quite fast enough for real time on the 133-MHz machine. When including the simulation core and other aspects of the simulation run cycle, this number dropped to around 15 frames per second. Average users running the simulation would have at least three times faster computers, so 15 frames per second on my machine wouldn't be a problem for them. In contrast, a hardware-rendered OpenGL grid landscape produced 4 frames per second on the same machine after substantial optimization.

Ocelot launched with a color map in the simulation; this method of viewing the simulation has increased its user base substantially (see Figure 5). Most users don't know Ocelot exists. They assume the smooth landscape graphics are rendered in hardware.

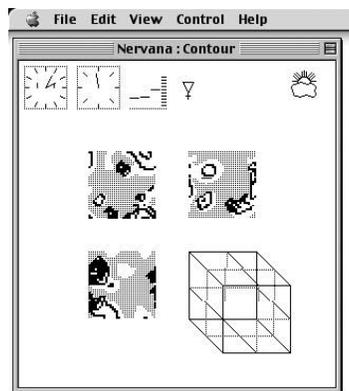


4 Diverse landscapes viewed through Ocelot.

5 Advantages of Ocelot versus the early vector graphics.



6 Early example of the brain viewed as a three-slice cut.



Brain in a box

Showing someone a landscape and having him or her quickly pick out visual features such as water, mountains, beaches, or grassland is relatively easy (see Figure 6). Having that person see a graphical representation of dreaming, panic, or hunger is more difficult. The problem of representing the internal cognitive simulation presented itself at the onset of development. I tested a number of methods for describing the brain states

graphically, using as my main criteria the desire to represent the cognitive simulation's 3D aspects while also emphasizing only the important information rather than overwhelming the user.

By early 2002, I had developed the rotating cube visualization method, which showed only the substantial changes in brain state. I developed it for a 1-bit window showing depth through rotation and increasing the point size as it passes through the screen. Interestingly, with this method I found myself very familiar with the different transient brain states that the apes went through. I could recognize subtle things such as when the ape was awakening or having a lucid dream. With the hundreds, if not thousands, of hours spent watching the simulation running, I was particularly surprised by how quickly novice users picked up the same points through the rotating cube graphical representation of the ape's brain state.

The cognitive simulation itself is a relatively substantial part of the time spent in the simulation core. It requires 32,768 (32 32 32) nonlinear calculations performed over 64 Kbytes of memory (see Figure 7). Optimizing the brain graphics was important so they too wouldn't consume so much time. Like the landscape rendering, the optimization came through parameter reduction and multiplication removal. I reworked the entire algorithm to make it integer-addition based. I also compressed the brain change information into packed bytes, reducing the amount of memory read when rendering the brain.

Guerilla science

The simulation has a relatively loyal, yet anonymous, user base. Each release receives between 4,000 and 6,000 downloads, most coming from Mac users, with Windows and Linux version users increasing with each release. At least a third of these downloads occur on release day. The releases are typically between two and six weeks apart,

although major changes have delayed some releases to 10 weeks. The general rule is eight releases per year.

To support the user base, I circulate a monthly email newsletter—*The Mailout*—and maintain a developer mailing list for those developing the simulation for particular platforms or with particular interests in the simulation. The Mailout has a historical connection with the development, as it is the primary means of communicating with dedicated simulation users and testers. The user base provides minimal feedback, leaving just occasional comments on download sites about new features they would like to see. For example, a recent request asked for scripting (high-level programming) in the simulation so users could program events to see how the apes would react, and then script one ape's behavior to see how other apes would react. Analysis showed that it would be relatively easy to add scripting later on, so I wrote a working document to introduce the idea. As part of a series of changes, I plan to introduce scripting in the next six months.

As one of my underlying objectives for the Noble Ape Simulation, I wanted to get the cognitive simulation out to a number of end users quickly—quite literally to get them thinking about thinking. This has been one of the Noble Ape's successes and the source of much of the feedback I've received. The underlying cognitive simulation method has undergone three major rewrites and is about to undergo a fourth. The simulation also lets others write their own cognitive simulation within the Noble Ape framework. Ideally, methods for nonprogrammers to access the cognitive simulation through scripting will let more people use the simulation as a tool.

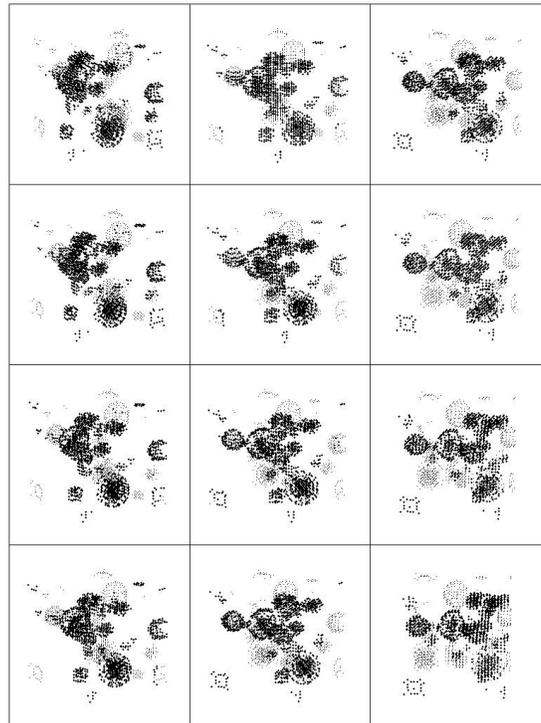
Stability

With so many users, and tens of thousands of hours spent running each simulation release, stability is critical. Stability here has two meanings—with the software and within the simulation.

In the simulation's initial development, I used memory conservatively. Memory-related crashes constitute a substantial portion of software bugs. While the early simulation had a relatively static approach to memory allocation, the modern simulation is more flexible with memory allocation.

To check the simulation core, a combination of command line tests operates through the simulation's command-line environment. As the simulation's graphics are also platform-agnostic, the command-line interface can test them as well. I check the command-line interface on Mac and Windows, too, to assure the platform-independent code returns the same results on both platforms. This kind of testing doesn't catch all bugs, but the short time between releases means that problems reported or found get fixed quickly.

Few parts of the simulation core serve just one purpose, with most code serving at least two functions. A number of the drawing functions have multiple purposes, while the movement code operates through a single function whether it's simulated movement or user-feedback movement—mouse-clicks in the map. This approach, adopted in the Stockholm Re-Write, also makes tracking bugs a little easier and improves the



7 Time slices of the rotating brain in a box.

code's maintainability and readability—two critical factors for Open Source development.

The simulation's stability was critical when it was first released and then retested in 2001. I also check stability conditions at least once per release to make sure the simulation doesn't "die out" for a novice user—the apes don't drown or starve.

As with the general software stability, testing the simulation stability occurs in the command-line environment. This environment lets users record particular aspects of the simulation at regular cycles and graph and analyze the output data. Hours of graphical simulation time can execute in minutes through the command-line environment interface.

Stability analysis in 2001 identified two main concerns:

- The biological simulation was too chaotic through seasonal fluctuations.
- The apes had long-term suicidal tendencies with the water surrounding the island. Most apes died from swimming out of their depth.

The biological simulation's quantum mechanics aspect gives population densities, but it doesn't give actual populations. Until 2001, the simulation used non-linear equations linked to the results of area integrals of the population densities to calculate the populations. I removed this part to allow for observed populations, which was the only important part of the biological simulation. At the time, this seemed like a short-term fix, although it persists to this day.

To solve the apes' morbid fascination with water, I rewrote the cognitive simulation code. This code has always involved a trade-off between idealism and hard-wiring. My long-term aim for the simulation is to remove

all hardwiring and let the apes form their own identities through the pure cognitive simulation. In 2001, the solution involved improving both the hardwiring and the resolution of the pure cognitive simulation. Like the biological simulation fix, this too is a work in progress. The main benefit of the Noble Ape development is that the timescale is so great that changes of this nature will take place. It's just a matter of waiting for the software development to reach the right phase.

Day-to-day development

Developing Noble Ape is not a full-time occupation. From the Stockholm analysis in 2001, I saw that there were central features in Noble Ape that could not be compromised if I wanted to sustain its future development. The commercial entities that others had proposed by then broke central tenets in the Noble Ape development. The only practical method going forward was to develop Noble Ape in the evenings as I had done in the earlier development. So, from mid-2001 to early 2002, I nailed down the Noble Ape development model.

This model allowed for development in 12 to 18 hours a week, which was equivalent to 18 to 24 hours a week because a number of additional hours were needed for nonkeyboard time. This division of keyboard and planning times improved the development greatly. My paid employment put my hands on a keyboard for at least nine hours a day. Spending an additional three hours a day with my hands on a keyboard wasn't always physically possible. So I used any free time to think about the problems currently encountered with the development.

I still use the weekends for developing large dedicated changes and devote weeknights to pruning changes and planning. I maintain a development log online, which provides relatively regular updates for people visiting the site and also maintains a good historical account of particular developments. This method trades speed of development for freedom and longevity. The latter two seem critical for a development like Noble Ape.

Practical Open Source

Open Source's romantic narrative has millions of developers working on and refining the source code provided. This hasn't been Noble Ape's history. For the Macintosh and Windows development and all the internal simulation core and simulation graphics, I am the only regular developer. One other developer has worked on the Linux port and two Apple Computer engineers volunteered Apple optimization code. Removing the romance, Open Source is about access. Open Source guarantees that I can continue to develop Noble Ape, as long as I have an Internet connection and a computer. Proprietary software without investment dies. Open Source software continues as long as people want to develop it.

I have a general rule of thumb: For every 1,000 downloads of the simulation, there will be one source code download. The average user doesn't care that the development is Open Source and about the source code; only a tiny fraction do. Open Source's benefits include an infrastructure, occasional community input, and a good industry understanding of Open Source.

Moving the development formally to an Open Source license in 2001 instantaneously gave me access to the substantial infrastructure freely available for Open Source development. Web hosting, free concurrent versions systems, mailing lists, and compiler farms are all available. Aside from giving you a computer and an Internet connection, almost everything comes with the package.

Open Source developers congregate on Open Source sites. If your development is relatively active, folks will visit the site and occasionally join in with discussions, offer advice, or ask for help. Similarly, doing releases through an Open Source release site is a good way to get the right people looking at the development.

When users come to your software on an Open Source site, they know what to expect. They understand where your software is practically and ethically and the level of commitment the software represents. As an early developer, I found it frustrating not to find good example code of particular operating system features I wanted to implement. Open Source also has an educational and mentoring purpose in providing functioning source code for new developers who want to implement particular features in their own code.

A recent success of Noble Ape's Open Source development has been its distribution by Apple Computers. Also, two Apple engineers contacted me about using the simulation in a display they were presenting at Apple's World Wide Developers' Conference 2003. Through this interaction, the Apple engineers provided a large section of G4/G5 optimized source code. In addition, the Apple version of the Noble Ape source code is distributed with Apple's Computer Hardware Understanding Development (CHUD) toolkit, so the simulation is included with Apple's developer tools.

Final motivation

I have two motivations for continuing the Noble Ape development. Online, I read how people use the simulation and how much they get out of it. I would like them to have a pleasurable, reflective experience with the simulation. Knowing about the experiences that people get from the simulation is a substantial motivation. In developing software, you sometimes lose sight of the end user's experience.

Actual progression is a second great motivator. There have been small breakthroughs with Noble Ape—things I've improved in the simulation's stability and, to a lesser extent, in the cognitive modeling. The development's Open Source nature lets others use the Noble Ape framework for their own work.

The greatest luxury with the development is time—time that has gone and has yet to go into the development. With current estimates and delta releases, the simulation will reach Version 1.0 in about 20 years. It could be sooner, but that time frame sits comfortably with me. There is still a lot to be done. ■

Readers may contact Tom Barbalet at tom@nobleape.com.

Contact editor Mike Potel at potel@wildcrest.com.